Storage Needs for GENIE

Gabriel Perdue
Fermilab
2014/September/3

GENIE

- **G**enerates **E**vents for **N**eutrino **I**nteraction **E**xperiments.
- http://genie.hepforge.org
- Well-engineered C++ software framework built on sound OO-principles and design patterns. (The Gang of Four is omnipresent.)
- Propagates a flux of neutrinos (specified by function, histogram, or ntuple) through a geometry (Geant4-compatible) and simulates the initial interaction and propagation of hard vertex products through the nuclear medium. Geant4 takes over when particles leave the nucleus.
- ROOT provides many core utilities. GENIE also heavily leverages other HEP and FOS software – LHAPDF, GSL, Pythia, log4cpp, etc.

Andreopoulos, C. and Bell, A. and Bhattacharya, D. and Cavanna, F. and Dobson, J. and others. "The GENIE Neutrino Monte Carlo Generator". Nucl.Instrum.Meth. A614. 87-104. 2010.

GENIE at FNAL

- GENIE is the primary event generator for:
  - ArgoNeut
  - LAr1-ND
  - LBNE
  - MicroBooNE
  - MINERvA
  - NOvA
- GENIE is also being considered for special studies by MINOS and MiniBooNE (they use previous generation software for their main generators).

Software Dependencies

- GENIE uses ROOT (5, eventually 6), Pythia(6, eventually 8), LHAPDF (5, eventually 6), log4cpp, GSL.
- Libraries: libstdc++, libc, libgcc, linux-vdso, libm, ld-linux-x86-64, libxml2 ... possibly not complete (ROOT, etc. have requirements).
- GENIE does not (yet) use any features from C++11.
- Generally, building on Scientific Linux is easy.
- Building 32-bit is also possible.

```
> ./cloc-1.60.pl R-2_8_0/
     3285 text files.
     3200 unique files.
     7197 files ignored.

http://cloc.sourceforge.net v 1.60  T=113.14 s (11.3 files/s, 4119.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                            525          30478          37587         176349
XML                            125          21895           2144         147176
C/C++ Header                   504           9052           8118          22282
Perl                            28            456           1469           3620
make                            47            514            485           1651
Bourne Shell                    34            157            334           1059
Bourne Again Shell               2            145            127            727
SQL                             12             37              0            117
-------------------------------------------------------------------------------
SUM:                          1277          62734          50264         352981
-------------------------------------------------------------------------------
```

There is a lot of configuration XML and experimental data packaged for the validation framework.

Development Needs

- GENIE requires ~250 MB (uncompressed) for its code and libraries, plus ~2 GB of third party code that will change rarely (85% is ROOT).
  - This may actually be more like ~10 GB if we keep many different versions of ROOT.
- Four core developers at the lab, plus 5-10 "moonlighters" from the experiments.
  - I keep, for example, almost 10 different versions of GENIE in my MINERvA working area.
- Storage requirements for development (not production!) is less than 100 GB per developer (justification follows).
- A "small" 5 TB BlueArc allocation would probably cover development needs for the foreseeable future.

Validation Needs

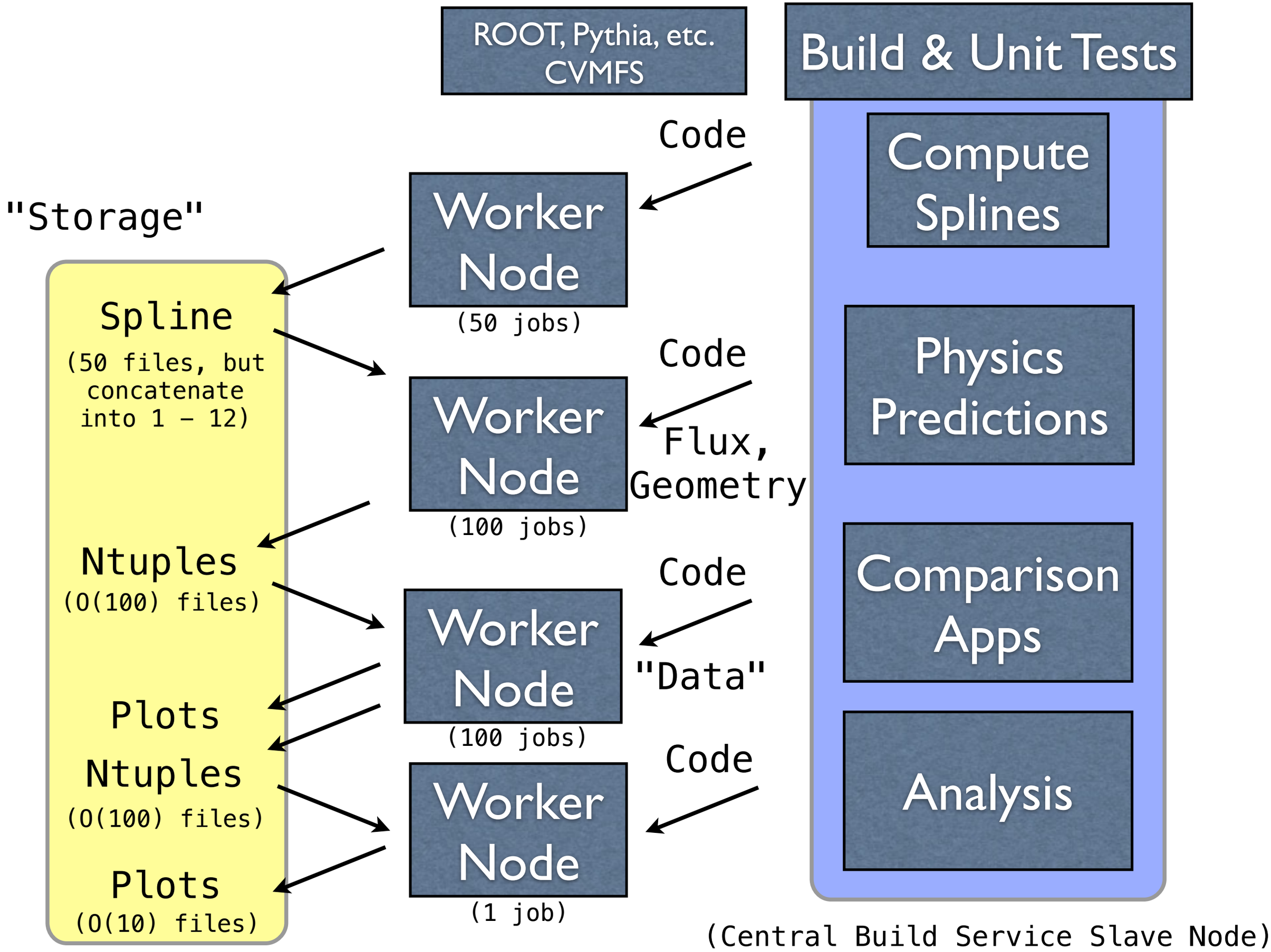
- Significantly more complicated storage picture.
  - Discuss "high-level" needs only here...
- The initial goal is to run the validation weekly.
- We would like to work towards a more rapid cycle, perhaps daily. This is less for development and more for...
- We would like to eventually be able to use these "validation runs" to do physics tuning.
  - This would require designing a compressed validation so that it could be run several times per day.

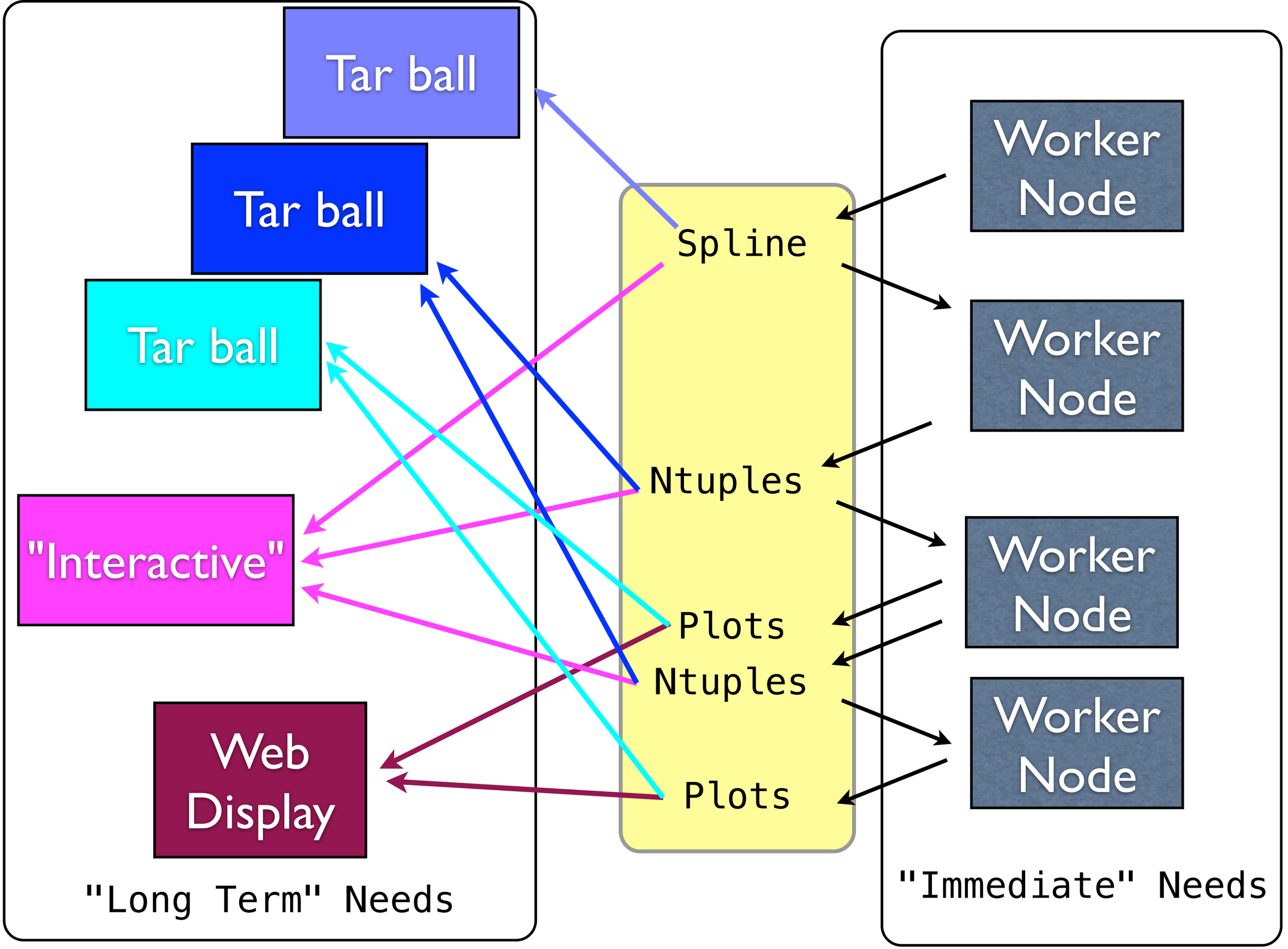
Scripting Framework

- Six Stages:
  - Build
  - Unit Tests (Eventually)
  - Generate Cross Sections (In: NA; Out: XML)
  - Generate Physics Predictions (In: XML, Geometry, Flux; Out: ROOT Ntuples)
  - Run Data/MC Comparison Apps (In ROOT Ntuples; Out ROOT Ntuples, PDFs)
  - Compare outputs to previous data/MC comparisons / Study global behavior (In ROOT Ntuples; Out ROOT Ntuples, PDFs)
- Each step depends on the previous step succeeding.
- We plan on using the Central Build Service to coordinate the flow from one stage to the next, but each stage will have its own script.
  - Working with LArSoft on this – there may be only one or two scripts, but instead each stage will be controlled by a different configuration file.

Scripting Framework

- Build
  - GENIE: 250 MB
  - 3rd Party (Pythia, LHAPDF, ROOT, GSL, log4cpp): 2 GB
    - Infrequent changes (so use CVMFS), but may want multiple copies of ROOT.
- Generate Cross Sections (In: NA; Out: XML)
  - Output size: ~1 GB (can vary by 1 order of magnitude based on number of targets, number of knots in the spline); Run time: ~12 hours per job (upper bound - light nuclei are much faster than heavy nuclei).
- Generate Physics Predictions & Run Data/MC Comparison Apps
  - Output size: ~100's of MB per App. Only a handful of apps on day one. (Several dozen are "available." - Plan to make many more.) Total storage: ~10 GB; Run time: ~several hours per job.
- Compare outputs to previous data/MC comparisons / Study global behavior (In ROOT Ntuples; Out ROOT Ntuples, PDFs)
  - Vaporware today... estimated output is small (summary plots must be human consumable).


ROOT, Pythia, etc.
CVMFS
Build & Unit Tests
Compute Splines
Physics Predictions
Comparison Apps
Analysis
Code
Code
Code
Flux, Geometry
"Data"
Worker Node
(50 jobs)
Worker Node
(100 jobs)
Worker Node
(100 jobs)
Worker Node
(1 job)
"Storage"
Spline
(50 files, but concatenate into 1 – 12)
Ntuples
(O(100) files)
Plots
Ntuples
(O(100) files)
Plots
(O(10) files)
(Central Build Service Slave Node)


Tar ball
Tar ball
Tar ball
"Interactive"
Web Display
"Long Term" Needs
Spline
Ntuples
Plots
Ntuples
Plots
Worker Node
Worker Node
Worker Node
Worker Node
"Immediate" Needs

Sizes

20 - 50 x ~20 MB

50 - 100 x ~ 100 MB

50 - 100 x ~5 MB??

50 - 100 x ~5 MB??

"Small"
~10 MB

Spline

Ntuples

Plots

Ntuples

Plots

Tabulated
Total Cross
Sections

"Physics
Predictions"
(Publication
Reproductions)

Generator-Data
Comparisons

Examine the
Comparisons

Spline Tar ball

20 - 50 x ~20 MB

ROOT Tar ball

50 - 100 x ~ 100 MB

Plots Tar ball

50 - 100 x ~5 MB??

"Interactive" (Spline, ROOT)

50 - 100 x ~5 MB??

Web (Plots)

"Small" ~10 MB

Spline Tar ball

1 GB Long Term

"Long Term" leaves some freedom for filtering.

ROOT Tar ball

10 GB Long Term

Plots Tar ball

1 GB Long Term

"Interactive" (Spline, ROOT)

11 GB Short Term (4-10 weeks)

Web (Plots)

1 GB "Long Term" (How far back REALLY needs to be visible?)

Aggregation and Tracking

- What is the aggregate request?
  - There is some flexibility – if there is less space we will filter older files more aggressively.
  - We want to keep *release* validation "forever" but this is small (100 GB per release is very conservative).
  - For tuning we may want to keep many dozens of copies of varied validation, but the timescale for retention would be short (weeks) and the long-term storage could be highly filtered and compressed.

- Suppose we wanted to keep 1000 validation cycles.
  - 10 TB
  - The main complications are:
    - File sizes for new validation apps could be quite large.
    - We may want to do many cycles when tuning.
    - Juggling files into the tape archive.

- Book-keeping
  - Ideally we heavily leverage SAM.
  - What do we need to do on the GENIE end to make this possible?

Timeline

- The goal is to complete the validation framework by next Summer.
- We will continue to develop validation and tuning applications for years, but the framework should be able to accommodate additional apps with no changes.

More Detailed Stage Discussion

Build

- Script in hand that does green-field builds with minimal checks for previously installed 3rd party codes.
- Currently assumes x86_64.
- Not CVMFS aware.
- Bash. Willing to rewrite into Python – it is missing some functionality anyway, e.g., how to declare files to SAM, etc.
  - Requires Git, wget, gcc 4.1+ (can probably go lower), gfortran, Python (2.6?) for the full stack (ROOT + Pythia6 + LHAPDF5).

Specify Config: GENIE, 3rd Party Code Versions

Checkout GENIE

Check for versioned 3rd Party Code
(in /grid/fermiapp or CVMFS)

Checkout 3rd Party Code

Build

Build GENIE

Declare binaries/
libraries to SAM?

Install GENIE – /grid/fermiapp/pds/genie

Declare data sets
/ support files
to SAM

```
                    Report Success 0
                  Failed Checkout 1
        Failed 3rd Party Build 2
         Failed 3rd Party Find 3
Failed 3rd Party Installation 4
           Failed GENIE Build 5
    Failed GENIE Installation 6
```

Generate Cross Sections

- "Bootstrap Option": Keep a set of splines in /pnfs/data(?) for testing other components of the scripting framework.
- Comparison: Store a complete set of splines from official releases, the last X validation releases (where X is probably one for automated comparisons).
- ~20-50 grid jobs if we do one job per target (could choose to break up the Event Generator Lists).
- Output at this stage is ~20-50 x ~20 MB files, which could be concatenated into a smaller set of files (or one file).
  - One for free nucleons, one for each element (favorites: He, C, O, N, Ar, Fe, etc. Minerva has ~20.).
- Jobs in the next stage must load the file, and the inefficiency in loading unneeded cross sections is outweighed by simpler coordination.

gmkspl

- **G**ENIE **M**a**k**e **Spl**ine
- Example:

```
$ gmkspl -p 14,-14 -t 1000060120 -o carbon_splines.xml \
--event-generator-list Default
```

- Set the Projectile (PDG code), the Target (PDG Ion Code), the output, and the Event Generator List.
- The `Default` list contains 13 generator lists (CCQE, COH, DIS, etc.).
- Each generator list contains from one to three (roughly) cross section models.

Example Cross Section Spline

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- generated by genie::XSecSplineList::SaveSplineList() -->

<genie_xsec_spline_list version="2.00" uselog="1">

<spline
   name="genie: LwlynSmithQELCCPXSec/Default/nu:-14;
   tgt:1000060120;
   N:2212;
   proc:Weak[CC],QES;"
   nknots="44">
<knot> <E>    0.01000 </E> <xsec>           0 </xsec> </knot>
<knot> <E>   0.030466 </E> <xsec>           0 </xsec> </knot>
<knot> <E>   0.050932 </E> <xsec>           0 </xsec> </knot>
<knot> <E>   0.071399 </E> <xsec>           0 </xsec> </knot>
<knot> <E>   0.091865 </E> <xsec>           0 </xsec> </knot>
<knot> <E>    0.11233 </E> <xsec>           0 </xsec> </knot>
<knot> <E>     0.1427 </E> <xsec> 8.487371629e-13 </xsec> </knot>
<knot> <E>    0.18129 </E> <xsec> 4.103713443e-12 </xsec> </knot>
<knot> <E>     0.2303 </E> <xsec> 8.367696679e-12 </xsec> </knot>
<knot> <E>    0.29257 </E> <xsec> 1.290439962e-11 </xsec> </knot>
<knot> <E>    0.37168 </E> <xsec> 1.807792243e-11 </xsec> </knot>
...
```

Cross Section Algorithm

Neutrino Flavor (PDG Code)

Target (Carbon)

Energy (GeV)

Cross Section (GeV^{-2})

Specify Config: List of targets, List of Event Generator Lists per target, code location

For each target material:

For each target Event Generator List:

Run `gmkspl` as a grid job.

Once all of the splines are finished, concatenate them, store the concatenated file in /pnfs(?); Declare concatenated output only to FTS/SAM?

```
                    Report Success 0
                Failed to find Code 1
Failed to concatenate and store 2
        Failed on Target <PDG Code>
```

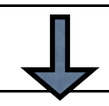
Generate Physics Predictions

- Each prediction is a unique snowflake.
- Requires the cross section spline and support files: a flux and target specification (likely to complicated, but small):
  - e.g., the NuMI flux on the MINERvA geometry
  - Sometimes very simple, e.g. 500 MeV electrons on carbon

Specify Config: List of predictions, code location, support files location paired with prediction (traditionally with the code); here each app will ask SAM for support files.

For each application:

Run its script (may be complicated, may be Bash, Python, Perl, etc.) as a grid job



As each of the apps are finished, store the output file in /pnfs(?) & Declare to FTS/SAM

```
                Report Success 0
      Failed to store output 1
Failed on finish app <App Code>
```

Comparison Applications

- Each application is a unique snowflake.
- Requires a published data set and the generator predictions file.
- Jobs should be very fast (unless they are doing a complicated fit, etc.).
- Output is plots and ROOT files (histograms and/or ntuples).

Specify Config: List of applications, paired list of data set and generator prediction files (each app will ask SAM for data set and prediction set)

For each application:

Run its script (may be complicated, may be Bash, Python, Perl, etc.) as a grid job

As each of the apps are finished, store the output file in /pnfs(?) Ntuples+Historgrams to BlueArc, Plots+ROOT files to /pnfs(?)

Report Success 0
Failed to finish app <App Code>